Solving Ordinary Differential Equation Based Constraints using the CP framework

Olivier MULLIER

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
101-8430 Tokyo – Japan
olivier.mullier@etu.univ-nantes.fr

Abstract. Numerous problems coupling constraints and ordinary differential equations (ODEs) rise in many different domains. We describe a natural and efficient way to make this kind of problem fit the numerical constraint satisfaction problem framework. In order to do it, using ODE solvers, we show how to slightly adapt some standard filtering algorithms from this framework in order to solve this kind of problem in a typical branch and prune algorithm. Some problems are presented as examples of applications.

1 Introduction

In a lot of domains, problems involving constraints with ordinary differential equations appear (kinematics modeling, dynamics of mechanical systems in engineering, biology, chemistry, and economics). Because these problems often contain some uncertainty, it is natural to apply interval techniques in order to model them. It is why many kinds of these problems have been treated (parameter estimation, control, design, proofs for chaotic dynamical systems) by specific methods using interval techniques (see for example [23, 10, 13, 12]) but this kind of method is dependant of the problem they solve. Because these problems involve variables belonging to continuous domains and having constraints on these, they fit the numerical CSP framework (NCSP). One just needs to express the ordinary differential equation of these problems as constraints and therefore gain one of the advantages of the CP framework that is being able to separate the problem declaration from the resolution process.

Some previous work ([4, 5]) has been able to put these problems into the NCSP framework. For that, a specific class of CSPs which includes ODE based constraints was designed : CSDP (Constraint Satisfaction Differential Problem). This class includes two kinds of variables : Some functional variables whose values are functions $\mathbf{x}(t)$ from $\mathbb{R}$ to $\mathbb{R}^n$ (corresponding to the trajectories of the ODE) and some real variables, called restriction variables, that correspond to characteristic values of functions $\mathbf{x}(t)$ (for example $\mathbf{x}(0)$ or $\max_{0\leq t\leq 1}\mathbf{x}(t)$). The constraints of the CSDP include constraints acting on the functional variables

(corresponding to the ODE definition) and constraints acting on the restriction variables (like the Value Restriction constraints, e.g. $||\mathbf{x}(0)|| \leq 1$, and the Maximum Restriction constraints, e.g. $\max_{0 \leq t \leq 1} \mathbf{x}(t) \leq 1$).

In this paper, we present a method that makes the ODE based constrained naturally fit the NCSP framework. This is done by abstracting an ODE by its *solution operator* which is a function that maps an initial condition and a time to the corresponding final state. In particular, this method is able to handle problems as parameter estimation problems and two-point boundary value problems to express them in the CP framework. Another requirement for this method is to slightly tune the standard filtering algorithms in order to solve efficiently these constraints. This is due to the expensiveness of the simulation of ODE required to evaluate the solution operator.

In section 2, we will start by bringing some notations and useful results from interval analysis used in this paper. We will see also notions about the NCSP framework to finish by introducing some ODE solvers from litterature and show briefly their resolution processes. Section 3 will be devoted to the method used in order to include ODE based constraints in the NCSP framework. In particular by defining the ODE solution operator and also by introducing some filtering algorithms where we will need to make some improvement. At the end implementation is presented to see exactly what kind of change has to be done. Section 4 is dedicated to some experiments made on several problems (academical one and also some from literature as well). Finally, conclusion and an outlook on future research is given in Section 5.

2 State of the art

In this section are introduced notions needed for the reading of this paper. We define some notations of interval analysis before overviewing the tools provided by the CP framework in order to handle NCSPs. Finally we make an introduction about ODE and the methods used by interval solvers to provide validated solutions.

2.1 Interval analysis

The birth of interval analysis is not dated with certitude but modern one appeared in the 60's due to Ramon E. Moore works ([16] and especially [17]). Interval analysis allows, not to manipulate approximated values of real numbers by machine representable numbers, but intervals where the values belong (for example, π can be defined by the interval $[3.14, 3.15]$).

Notations In this paper and as seen above, intervals will be denoted by square brackets : $[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} |\ \underline{x} \leq x \leq \overline{x}\}$ is an interval where $\underline{x}$ and $\overline{x}$ are respectively lower bound and upper bound of the interval. $\mathbb{IR}$ will be the set of closed intervals :

$$\mathbb{IR} = \{[x] = [\underline{x}, \overline{x}] \ |\ \underline{x}, \overline{x} \in \mathbb{R}, \underline{x} \leq \bar{x}\}$$

For some interval $[a]$, quantities are also defined :

- width of $[a]$: $\text{wid}(a) = \overline{a} - \underline{a}$;
- midpoint of $[a]$: $\text{mid}(a) = (\overline{a} + \underline{a})/2$;

Interval vectors are defined as vector with interval components and will be denoted by bold symbols ($[\mathbf{x}]$ is an interval vector also called box). In this case, we can still define the midpoint for $[\mathbf{a}] \in \mathbb{IR}^n$:

- midpoint of $[\mathbf{a}]$: $\text{mid}([\mathbf{a}]) = \begin{pmatrix} \text{mid}([\mathbf{a}]_1) \\ \text{mid}([\mathbf{a}]_2) \\ \vdots \\ \text{mid}([\mathbf{a}]_n) \end{pmatrix}$

Definition 1 (Interval Hull). *Let S be a relation of $\mathbb{R}^n$. The hull of S (denoted $\Box S$) is defined as the box in $\mathbb{IR}^n$ verifying :*

1. $S \subseteq \Box S$;
2. $\forall [\mathbf{x}] \in \mathbb{IR}^n$, $S \subseteq [\mathbf{x}] \Rightarrow \Box S \subseteq [x]$.

Interval Arithmetic Operations. Real based arithmetics can be easily extended to intervals as follows : let $[a]$ and $[b] \in \mathbb{IR}$ and $\diamond \in \{+, -, \times, /\}$. Interval-arithmetics are defined by

$$[a] \diamond [b] = \Box\{x \diamond y \mid x \in [a], y \in [b]\},\ 0 \notin [b] \text{ when } \diamond = / \tag{1}$$

For each operator ($\times$ is omitted in the notation), we can define its proper calculation algorithm :

$$\begin{aligned} [a] + [b] &= \left[\underline{a} + \underline{b},\ \overline{a} + \overline{b}\right] \\ [a] - [b] &= \left[\underline{a} - \overline{b},\ \overline{a} - \underline{b}\right] \\ [a]\,[b] &= \left[\min\{\underline{ab},\ \underline{a}\overline{b},\ \overline{a}\underline{b},\ \overline{a}\overline{b}\}, \max\{\underline{ab},\ \underline{a}\overline{b},\ \overline{a}\underline{b},\ \overline{a}\overline{b}\}\right] \\ [a]\,/\,[b] &= [\underline{a}, \overline{a}] \times \left[1/\overline{b},\ 1/\underline{b}\right],\ 0 \notin [b] \end{aligned}$$

The extended interval division. As we can see the calculation algorithm for the interval division can lead to an issue if 0 belongs to the denominator. This can be improved as follows :

$$0 \in [x] \in \mathbb{IR} \Rightarrow \frac{1}{[x]} = \left[-\infty,\ \frac{1}{\underline{x}}\right] \cup \left[\frac{1}{\overline{x}},\ +\infty\right]$$

The extended interval division is useful when inclusion of the result has to be done (As for example in the multidimensional interval Newton method but it will be shown later in Section 3.3).

Algebraic Properties. Interval operations do not have all the same properties as real operations as one could expect. Addition and multiplication remain associative and commutative. Substraction is not the inverse of the addition : $[a] + [b] = [c] \nRightarrow [a] = [b] - [c]$ in general. The same observation can be made for the division. All arithmetic operations are inclusion monotone : let $[A]$, $[B]$, $[a]$ and $[b] \in \mathbb{IR}$ such that $[a] \subseteq [A]$ and $[b] \subseteq [B]$ then $[a] \diamond [b] \subseteq [A] \diamond [B]$ with $\diamond \in \{+, -, \times, /\}$.

Interval-valued Functions. We can extend the definition (1) for elementary functions (exp, ln, sin, cos, etc.) as well. Finding the implementation for such functions generally needs to study their monotonic parts. Let $f : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}$ be an elementary function :

$$f([x]) = \square\{f(x)|\ x \in [x]\} \tag{2}$$

We can define the interval extension for functions being expression compound of these elementary operations.

Definition 2. *A function* $[f] : \mathbb{IR} \to \mathbb{IR}$ *is an interval extension of* $f : \mathbb{R} \to \mathbb{R}$ *if :*

$$\forall\, [x] \in \mathbb{IR},\ \{f(x)|x \in \mathbb{R} \cap [x]\} \subseteq [f]\,([x])$$

Remark 1. Interval extension for n-ary functions and function vectors can be defined as well.

Definition 3 (Natural Interval Extension).
Let $f : \mathbb{R}^n \to \mathbb{R}$ *be a continuous function on* $D \in \mathbb{R}^n$*. The natural interval extension of* f *is obtained by replacing each occurrence of a real variable by the corresponding interval.*

Example 1. Let $f(x) = x - x$ *and an interval* $[x] = [1,\ 2]$.

$$\Rightarrow [f]\,([x]) = [f]\,([1,\ 2]) = [1,\ 2] - [1,\ 2] = [-1,\ 1]$$

Theorem 1 (Fundamental Theorem of Interval Analysis [21]). *Evaluation of a natural interval extension of an expression gives rise to an interval enclosure of this function (Cf Example 1) :*

$$\{f(x)|\ x \in [x]\} \subseteq [f]\,([x])$$

Two phenomena explain this overestimation : the wrapping effect and the data dependency. The wrapping effect[11] is due to the natural interval extension that must results to the hull of $\{f(x)|\ x \in [x]\}$. The data dependency rises when we have multiple occurences in the evaluation of f (Cf. Example 1 : $[f]\,([x]) = [-1,\ 1]$ while $\{f(x)|\ x \in [x]\} = \{0\}$).

Theorem 2 (Moore's Theorem[17]). *Let* $f(x) : D \to \mathbb{R}$ *be given by a real expression having only one occurence of* x*. Then,*

$$\forall\, [x] \in \mathbb{IR} : \square\{f(x)|\ x \in [x]\} = [f]\,([x])$$

Definition 4 (Mean Value Extension). *Let $f : D \subseteq \mathbb{R}$ be continuously differentiable. Let $[x] \subseteq D$ and let $c \in [x]$. Then the mean value extension of f over $[x]$ and centered at c is defined by*

$$[f]([x], c) = f(c) + [f']([x])([x] - c)$$

Another example of extension is the Taylor interval one. It computes a high order polynomial approximation of the Taylor series expansion of the function (see [17]).

Definition 5 (Taylor Interval Extensions[17]). *Let $f : D \to \mathbb{R}$ be a class $\mathbf{C}^{n+1}$ real function. The n^{th} order Taylor interval extension of f over an interval $[x] \subseteq D$ and centered at $c \in [x]$ is defined by*

$$[f]([x], c) = \sum_{i=0}^{n-1} \frac{f^{(i)}(c)}{i!}([x] - c)^i + \frac{\left[f^{(n)}\right]([x])}{n!}([x] - c)^n$$

Remark 2. The mean value extension is equivalent to the first order Taylor extension.

2.2 Numerical Constraint Satisfaction Problems

Numerical Constraint Satisfaction Problems (NCSPs)[6] are CSPs with constraints taking values over continuous domains.

Definition 6 (Numerical Constraint Satisfaction Problem (NCSP)). *An NCSP $P = (\mathcal{V}, \mathcal{C}, \mathcal{D})$ is defined by a set of variables $\mathcal{V} = (x_1, ..., x_n)$ taking their values in continuous domains $\mathcal{D} = (D_1, ..., D_n)$ and $\mathcal{C} = \{C_1, ..., C_n\}$ is a set of constraints. $\mathcal{V}$ can be seen as a vector $\mathbf{x} = (x_1, ..., x_n)$ and $\mathcal{D}$ as a box $[\mathbf{x}]$.*

Definition 7 (Solution of a NCSP). *A solution of a NCSP P is an instantiation of the variables of $\mathcal{V}$ belonging in their domains of $\mathcal{D}$ and that verify the set of constraints $\mathcal{C}$. The set of solutions is denoted by* $\mathrm{Sol}(P)$.

$$\mathbf{a} \in \mathrm{Sol}(P) \Leftrightarrow \mathbf{a} \in [\mathbf{x}] \text{ and } \big(\forall c \in \mathcal{C}, c(\mathbf{a})\big)$$

To solve a NCSP, a way is to apply the branch and prune algorithm (see e. g. [9] and references therein). This algorithm alternates branching over the domains and pruning them using the constraints. The branch and prune algorithm for NCSP (cf algorithm 1, page 6) produces two pavings (sets of boxes) $\mathcal{B}$ the *boundary boxes* and $\mathcal{S}$ the *solution boxes*. The algorithm is a skeleton that can be applied to many specific requirements. It is the function $\mathrm{Contract}_{\mathcal{C}}$ that is dependant of the problem and will prune the domain.

Definition 8. $\mathrm{Contract}_{\mathcal{C}}$ *Let* $\mathbf{x} \in [\mathbf{x}] \subseteq \mathcal{D}$.

$$\big(\forall c \in \mathcal{C}, c(\mathbf{x})\big) \Longrightarrow \mathbf{x} \in \mathrm{Contract}_{\mathcal{C}}([\mathbf{x}]).$$

Algorithm 1: Branch and Prune Algorithm.

Input: $\mathcal{C} = \{c_1, \ldots, c_m\}$, $[\mathbf{x}] \in \mathbb{IR}^n$, $\epsilon > 0$
Output: $\mathcal{S} \subseteq \mathbb{IR}^n$, $\mathcal{B} \subseteq \mathbb{IR}^n$
$\mathcal{L} \leftarrow \{[\mathbf{x}]\}$; $\mathcal{B} \leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$;
while $\mathcal{L} \neq \emptyset$ **do**
  $([\mathbf{x}], \mathcal{L}) \leftarrow \text{Extract}(\mathcal{L})$;
  if $\text{IsSolution}_{\mathcal{C}}([\mathbf{x}])$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{[\mathbf{x}]\}$;
  else if $\text{wid}([\mathbf{x}]) < \epsilon$ **then** $\mathcal{B} \leftarrow \mathcal{B} \cup \{[\mathbf{x}]\}$;
  else
    $[\mathbf{x}] \leftarrow \text{Contract}_{\mathcal{C}}([\mathbf{x}])$;
    if $[\mathbf{x}] \neq \emptyset$ **then**
      $\{[\mathbf{x}'], [\mathbf{x}'']\} \leftarrow \text{Split}([\mathbf{x}])$;
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{[\mathbf{x}'], [\mathbf{x}'']\}$;
    end
  end
end
return $(\mathcal{S}, \mathcal{B})$;

Remark 3. When $\text{Contract}_{\mathcal{C}}$ contracts a box, no solution of the NCSP is lost and all the solutions are eventually contained in one box from $\mathcal{B}$ or $\mathcal{S}$.

$$\text{Sol}(\mathcal{P}) \subseteq (\cup\mathcal{S}) \cup (\cup\mathcal{B}) \tag{3}$$

Definition 9. $\text{IsSolution}_{\mathcal{C}}$ $\text{IsSolution}_{\mathcal{C}}$ *is used to compute the set of pavings* $\mathcal{S}$*. A box belongs to* $\mathcal{S}$ *if* $\text{IsSolution}_{\mathcal{C}}$ *returns true. The semantic of this function depends on the problem that is treated. Two kind of problems will be treated here : the conjunction of inequality constraints and well constrained systems of equations.*

Conjunction of inequality constraints In this case only inequality constraints occur. It is convenient to group them into one inequality constraint involving a vector valued function :

$$f : \mathbb{R}^n \to \mathbb{R}^m, \ f(\mathbf{x}) \leq 0$$

$\text{IsSolution}_{\mathcal{C}}$ returns $\begin{cases} \text{true if } [f]([\mathbf{x}]) \leq 0 \\ \text{false otherwise.} \end{cases}$ In this case, $\cup\mathcal{S} \subseteq \text{Sol}(\mathcal{P})$.

Well constrained systems of equations This involves n equations and n variables with possibly inequality constraints. The constraint can be written as this :

$$f(\mathbf{x}) = 0, \forall \mathbf{x} \in \mathcal{D} \tag{4}$$

$\text{IsSolution}_{\mathcal{C}}$ returns true if the existence of one unique solution in $[\mathbf{x}]$ has been proved.

2.3 Rigorous ODE solving

Solving ODE using interval techniques presents two advantages : when a solution is provided, the problem treated is guaranted to have a unique solution and an enclosure of this solution is given. Several techniques have been introduced in the past for solving ODEs.

Definition 10 (IVP). *A classical first order ODE can be written as follows :*

$$\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t)) \tag{5}$$

where $\mathbf{h} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ *is a vector field of class* C^1 *and* $t \in \mathbb{R}$.

The solution usually contains a free parameter so in order to find a solution, one must determine an initial value condition like this :

$$\mathbf{x}(t_0) = \mathbf{x}_0,\ t_0 \in \mathbb{R}$$

Then it is called an initial value problem (IVP). Since there is often uncertainty in the initial condition when we try to manage real problems, the initial condition is more likely described as an interval vector :

$$\mathbf{x}(t_0) \in [\mathbf{x}_0]\,,\ t_0 \in \mathbb{R}$$

After an integration, (5) can be rewritten as follows :

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \int_0^t \mathbf{h}(\mathbf{x}(\tau))d\tau \tag{6}$$

We can replace the integral by a conservative approximation :

$$\int_0^t \mathbf{h}(\mathbf{x}(\tau))d\tau \subseteq [0,t]\mathbf{h}([B]) \tag{7}$$

where $[B]$ is a bounding box enclosing all reachable states in the interval $[0,t]$. From (6) and (7), we obtain an interval enclosure of $\mathbf{x}$:

$$\mathbf{x}(t) \subseteq \mathbf{x}(t_0) + [0,t]\mathbf{h}([B]) \tag{8}$$

Several validated solvers for computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation have been implemented. For example :

VNODE-LP[19] (Validated Numerical ODE)
ValEncIA-IVP[24] (VALidation of state ENCLosures unsing Interval Arithmetic for Initial Value Problems)
CAPD[3] (Computer Assisted Proof in Dynamics)

Most of solvers are based on Taylor extension ([7, 15, 25, 2]). The algorithm for these solvers follow the same skeleton. It can be split into two steps :

1. Validate existence and uniqueness and then provide an a priori enclosure of the solution :
   – VNODE-LP : $[B]$ in (8) is computed using the Picard-Lindelöf operator[1] (see e.g. [22]) and the Banach fixed-point theorem (see for example [27]) that provide the uniqueness and validation of the solution computed :

$$[B^{(K+1)}] = [x^0] + [0,t]f([B^{(K)}]) \tag{9}$$

   with $[B^{(0)}] = [x^0]$.
   While $[B^{(K+1)}] \not\subseteq [B^{(K)}]$, $[B^{(K)}]$ is enlarged. If $[B^{(K+1)}] \subseteq [B^{(K)}]$, Picard is iterated until $[B^{(K+1)}] \approx [B^{(K)}]$. If the iteration does not converge or $[B^{(K+1)}]$ become too large, the time t is reduced. More details can be found in [19]
   – ValEncIA-IVP: $[B]$ in (7) is replaced by $[x_{encl}(t)]$ where :

$$[x_{encl}(t)] = x_{app}(t) + [R(t)] \tag{10}$$

   $x_{app}(t)$ is an approximate solution of the IVP and $[R(t)]$ is an interval containing the unknown error terms. There are two ways to calculate the approximate solution $x_{app}(t)$: analytically using an easy-to-solve reference system (by linearization generally) and numerically using non-validated numerical techniques for the solution of IVP. In order to make Valencia-IVP a validated solver, $[R(t)]$ must contains errors terms. The proof of the validation for Valencia-IVP is in [1].
2. Compute tighter enclosure : Goal is to use the a priori enclosure to perform a tighter enclosure. The most used approach is based on Taylor expansion coupling with the mean value extension. To minimize the wrapping effect, other techniques are performed (as Lohner's QR factorization, see [18] and reference therein).

3 Ordinary Differential Equations Based Constraints in the Standard CP framework

In order to use filtering algorithms that will prune the domains, we need to make the ODE fit the CP framework. This work must be done in both sides: ODEs need to be managed to fix the NCSP framework when filtering algorithms also need to take account of difficulties rising from simulating ODE. This Section will first describe the ODE solution operator and filtering algorithms that are designed for precised kinds of problems.

3.1 The ODE Solution Operator and its Derivatives

For a given ODE defined as above (section 2.3) that maps an initial condition : $\mathbf{x}(t_0) \in \mathbb{R}^n$ and a duration $t \in \mathbb{R}$ to a unique vector $\mathbf{x}(t)$ (proven by the ODE solver that is used), we are able to construct the ODE solution operator.

[1] also called Cauchy-Lipschitz in French.

Definition 11 (ODE Solution Operator). *Let $\Phi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ characterized by*

$$\Phi(\mathbf{x}(t_0), t) = \mathbf{x}(t_0 + t) \tag{11}$$

is called an ODE solution operator.

The role of this ODE solution operator is to abstract the simulation of the ODE from the CP framework point of view and to allow to manipulate ODEs like any other functions. Since the filtering algorithms introduced need the Jacobian $D\Phi$ of the function to be applied, we define it also :

Let $D\Phi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n\times(n+1)}$ be the Jacobian of Φ.

$$D\Phi = (D_{\mathbf{x}}\Phi \mid D_t\Phi) \tag{12}$$

with $D_{\mathbf{x}}\Phi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^{n\times n}$ being the ODE first variational equation, which is a linear non-autonomous ODE of dimension n^2 and $D_t\Phi : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^{1\times n} = \mathbf{h}(\Phi(\mathbf{x}, t))$. It results that evaluating Φ and $D\Phi$ is equivalent as solving an ODE of dimension $n^2 + n$. Since computation of ODE by ODE solvers is really expensive in term of time, the evaluation of this ODE solution operator should be as less as possible. It is why filtering algorithms have to be slightly modified to take into account this kind of difficulty.

3.2 Filtering algorithms

Filtering algorithms used depend on which kind of problem has to be solved. We need so to define them for each kind of problems and also show what kind of changes has to be done during the implementation of this method.

Conjunction of inequality constraints Two filtering algorithms are used for this kind of problems.

The first one is a simple interval enclosure test that consists of computing $[f]([\mathbf{x}])$ and see if all the box can be rejected. Let $[\mathbf{x}]$ be a box, $[\mathbf{x}]$ can be rejected regarding the constraint $f(\mathbf{x}) \leq 0$ if $\underline{[f]([\mathbf{x}])} > 0$ (where here 0 means the vector as same dimension as $f(\mathbf{x})$ and having only 0 for components).

Remark 4. It is important to notice that, because $[f]([\mathbf{x}])$ leads to an overestimation of the range of f, having $\underline{[f]([\mathbf{x}])} \leq 0$ does not mean for sure that the initial box $[\mathbf{x}]$ contains solutions.

The second filtering algorithm is the unidimensional interval Newton that allows to remove a slice of $[\mathbf{x}]$ as follows : the domain $[x_j]$ of the variable x_j is contracted to the new domain $\left[x'_j\right]$ applying the unidimensional interval Newton for each component of $f(\mathbf{x}) \leq 0$:

$$[x'_j] = \bigcap_{i\in\{1,\ldots,m\}} \left(\tilde{x}_j - \frac{1}{[a_{ij}]}\left([b_i] + [0,\infty] + \sum_{k\neq j}[a_{ik}]([x_k] - \tilde{x}_k)\right)\right) \cap [x_j] \tag{13}$$

where $[A] = [D\mathbf{f}]([\mathbf{x}])$, $[\mathbf{b}] = [\mathbf{f}](\tilde{\mathbf{x}})$ for some $\tilde{\mathbf{x}} \in [\mathbf{x}]$, usually the midpoint of $[\mathbf{x}]$. The addition of the interval $[0, \infty]$ allows applying the interval Newton, which is defined for equality constraints, to inequality constraints: Indeed, $f(x) \leq 0$ is equivalent to $0 \in f(x) + [0, \infty]$ and the interval Newton for equality constraint is actually applied to the latter.

Well constrained systems of equations Here also, two filtering algorithms can be used. In parallel with the simple interval enclosure from conjunction of inequality constraints, the constraint $f(\mathbf{x}) = 0$ implies that if 0 does not belong to the interval extension $[f]([\mathbf{x}])$, that means that the box $\mathbf{x}$ can be rejected.

The second filtering algorithm is the multidimensional interval Newton (the Krawczyck version[21] where some preconditionning is performed). It can contract the domain $[\mathbf{x}]$ to the new domain $[\mathbf{x}']$ as follows:

$$[\mathbf{x}'] = \big((I - C[A])[\mathbf{x}] + C[\mathbf{b}]\big) \cap [\mathbf{x}] \tag{14}$$

where $[A]$ and $[\mathbf{b}]$ are defined like in (13), and $C = (\text{mid}[A])^{-1}$.

Remark 5. In the branch and prune algorithm for well constrained systems of equations, the multidimensional interval Newton role plays $\text{Contract}_{\mathcal{C}}$ and $\text{IsSolution}_{\mathcal{C}}$ as well since this filtering algorithm is able to prove the existence and unicity of solution (again, see [21]).

3.3 Implementation

The implementation of this method made for the experiments has been created in C++ with the use of Profil/Bias[14] for interval computation. For the ODE solver, because we need evaluations of f over $[\mathbf{x}]$, $\mathbf{x} \in [\mathbf{x}]$ and $[Df]$ over the whole domain, our choice went to CAPD that provides all the requirements in order to apply the filtering algorithms.

Remark 6. In CAPD, computation of $f(\text{mid}([\mathbf{x}]))$ requires a separated simulation of the ODE when the one for $[f](\text{mid}([\mathbf{x}]))$ and $[Df](\text{mid}([\mathbf{x}]))$ are performed at once.

Unidimensional interval Newton This algorithm has been implemented in order to manage constraints of type $f(\mathbf{x}) \in [\mathbf{y}]$.

For saving computational time, $f(\tilde{\mathbf{x}})$ and $[Df]([\mathbf{x}])$ are not calculated at each improvement made on one component of $\mathbf{x}$. The advantage of this is that ODE solver will be called less time, saving the computational time. But it means also that the pruning will be less efficient than applying the regular unidimensional Newton method.

Algorithm 2: Unidimensional Interval Newton algorithm.

Input: function $f : \mathbb{R}^n \to \mathbb{R}^m$, box $[\mathbf{x}]$, interval matrix $[Df]([\mathbf{x}])$, box $[\mathbf{y}]$
Output: new contracted box $[\mathbf{x}]$
$\tilde{\mathbf{x}} \leftarrow \text{mid}([\mathbf{x}])$;
for *int i from 1 to n* **do**
  for *int j from 1 to m* **do**
    $[f_j] \leftarrow f(\tilde{\mathbf{x}})_j + \sum_{k \neq j} [Df]([\mathbf{x}])_{kj}([\mathbf{x}_k] - \tilde{\mathbf{x}_k})$;
    $[\mathbf{x}_i] \leftarrow [\mathbf{x}_i] \cap (([\mathbf{x}_i] - \tilde{\mathbf{x}}_i \cap -\frac{[f_j]-[y_j]}{[Df]([\mathbf{x}])_{ij}}) + \tilde{\mathbf{x}}_i)$;
    if $[\mathbf{x}_i] = \varnothing$ **then**
      return False;
    end
    else
      return True
    end
  end
end

Algorithm 3: Gauss Seidel algorithm.

Input: Interval matrix $[A]$, box $[\mathbf{x}]$, box $[b]$
Output: new contracted box $[\mathbf{x}]$
for *all components* $[\mathbf{x}]_i$ *of* $[\mathbf{x}]$ **do**
  $[\mathbf{x}]_i \leftarrow [\mathbf{x}]_i \cap \frac{[b]_i - [A]_i \times [\mathbf{x}]}{[A]_i i}$;
  if $[\mathbf{x}]_i = \varnothing$ **then return** False;
end
return True

Multidimensional interval Newton This uses the Gauss Seidel method made to solve linear systems of equations of the form $[A] \times [\mathbf{x}] = [b]$ (Cf. Algorithm 3).

An improvement of the method is to precondition the system by multipliying each member of the system of equation so that the problem become easier to solve. The solutions of $[A]\,[\mathbf{x}] = [b]$ are similar to a second system of the form $P\,[A]\,[\mathbf{x}] = P\,[b]$. P is chosen so that $P\,[A]$ approximates the diagonal matrix (P should be then an inverse of $[A] : [A]^{-1}$). This improvement is usually made by computing the inverse of $Df(\text{mid}([\mathbf{x}]))$ but, because it requires the computation of it, we use instead the inverse of $[Df]\,([\mathbf{x}])$. It means the approximation of the diagonal matrix will be poorer but saves computation time.

Algorithm 4: Multidimensional interval Newton algorithm.

Input: function $f : \mathbb{R}^n \to \mathbb{R}^m$, box $[\mathbf{x}]$
Output: new contracted box $[\mathbf{x}]$
boolean stop $\leftarrow$ False;
while *not stop* **do**
  $[b] \leftarrow [Df]\,([\mathbf{x}])^{-1} \times -f(\text{mid}([\mathbf{x}]))$;
  $[A] \leftarrow [Df]\,([\mathbf{x}])^{-1} \times Df([\mathbf{x}])$;
  $[\mathbf{y}] \leftarrow [\mathbf{x}] - \text{mid}([\mathbf{x}])$;
  if *Gauss_Seidel*$([A], [y], [b])$ *and* $[\mathbf{x}] \cap ([\mathbf{y}] + mid([\mathbf{x}])) \neq \varnothing$ **then**
    stop $\leftarrow$ there is no improvement;
  end
end

Remark 7. We can see that all these filtering algorithms are independent of the nature of f and can be used in order to solve constraints that do not involve ODEs.

3.4 Treated problems : parameter estimation problems

In parameter estimation problem, given some constraints, one needs to find the initial conditions of an IVP that satisfy these constraints. In real problems, these initial conditions are oftently given by measurements or observations, it is why intervals fit well to this kind of problem. A parameter estimation problem comes with an ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$ and measurements $(t_i, [\mathbf{m}_i])$ for $i \in \{1, \ldots, p\}$ that dictates the possible trajectories the ODE can have. A solution is then to find initial conditions for the ODE that satisfies

$$\mathbf{x}(t_i) \in [\mathbf{m}_i] \text{ for } i \in \{1, \ldots, p\} \tag{15}$$

Seen through the NCSP framework, a parameter estimation problem P = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ with

– $\mathcal{V} = \mathbf{x} = (x_1, \ldots, x_n)$ the initial conditions we are looking for:
– $\mathcal{D} = [\mathbf{x}] = ([\mathbf{x}_1], \ldots, [\mathbf{x}_n])$ is the initial search space for these conditions;
– $\mathcal{C}$ are the constraints that dictate the possible trajectories of the ODE with

$$\mathcal{C} = \{\Phi(\mathbf{x}, t_1) \in [\mathbf{m}_1], \ldots, \Phi(\mathbf{x}, t_p) \in [\mathbf{m}_p]\}. \tag{16}$$

All constraints are inequality ones so this kind of problem can be solved using the evaluation test and the unidimensional interval operator as explained in Section 3.2.

To save computational time, a special care has to be made in the constraints. Each constraint c_i require to simulate the ODE until time t_i and so must not to be treated independently. Sorting the constraints by increasing time will allow to perform only one simulation of the ODE for all measurements. Moreover, simulating ODE with large intervals can lead to an issue that make the ODE solver unable to reach the last time t_i for a given box.

An other improvement is then to keep the results that has been calculated so far and to replace the missing one by the interval $[-\infty, \infty]$. This way the filtering algorithm employed will be still able to contract some variables that could be calculated (and then reduce the size of the treated box).

3.5 Treated problems : Two-point boundary value problems

In the Two-Point Boundary Value Problem (TPBVP) we still are given an ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$ and n equality constraints g_i on $\mathbf{x}(t_0)$ and $\mathbf{x}(t_1)$. There we need to find trajectories of the ODE that will satisfy the ODE and the constraints $g_i(\mathbf{x}(t_0), \mathbf{x}(t_1)) = 0$. As the trajectory is completely defined by its initial condition, the TPBVP actually consists of finding the initial conditions that give rise to such trajectories.

Such a problem perfectly fits the NCSP framework since these initial conditions are the solutions of the NCSP (V, D, C) where:

– $\mathcal{V} = \mathbf{x} = (x_1, \ldots, x_n)$ are the initial conditions we search;
– $\mathcal{D} = [\mathbf{x}] = ([\mathbf{x}_1], \ldots, [\mathbf{x}_n])$ is the initial search region for the initial conditions
– $\mathcal{C} = \{c_1(\mathbf{x}) = 0, \ldots, c_n(\mathbf{x}) = 0\}$ are the constraints with

$$c_i(\mathbf{x}) = g_i(\mathbf{x}, \Phi(\mathbf{x}, t_1 - t_0)) \tag{17}$$

Note that once $[\Phi]([\mathbf{x}], t_1 - t_0)$, $[\Phi](\text{mid}[\mathbf{x}], t_1 - t_0)$ and $[D\Phi]([\mathbf{x}], t_1 - t_0)$ are evaluated then $[c_i]([\mathbf{x}])$, $[c_i](\text{mid}[\mathbf{x}])$ and $[Dc_i]([\mathbf{x}])$ directly follow from (17). In particular using the chain rule we obtain

$$Dc_i(\mathbf{x}) = \left.\frac{\partial g_i(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u}}\right|_{(\mathbf{x},\mathbf{y})} + \left.\frac{\partial g_i(\mathbf{u}, \mathbf{v})}{\partial \mathbf{v}}\right|_{(\mathbf{x},\mathbf{y})} D_{\mathbf{x}}\Phi(\mathbf{x}, t_1 - t_0) \tag{18}$$

with $\mathbf{y} = \Phi(\mathbf{x}, t_1 - t_0)$. The expression (18) can be evaluated for interval arguments using $[\Phi]([\mathbf{x}])$ and $[D\Phi]([\mathbf{x}])$.

4 Experiments

In this section, after having made some ODE solver comparisons, we study different examples of the introduced problems to illustrate with computational time and solution analysis the behaviour of our method. The algorithm has been written in C++, CAPD library was used to integrate the ODE system in each problem. All problems were solved on an Intel Pentium SU4100 1.3 GHz machine running on Linux.

4.1 ODE solver comparison

Let define an example of ODE to simulate that will allow to compare results from different solvers. For that we chose to simulate the Van Der Pol Oscillator whose van der Pol introduced as a mathematical model of self-sustained oscillations of a triode circuit with a cubic current-voltage characteristic [26]. This oscillator evolves in time according to the second order differential equation :

$$x''(t) = \mu(1 - (x(t))^2)x'(t) + x(t) \tag{19}$$

that can be rewritten as :

$$\begin{cases} x_1'(t) = x_2(t) \\ x_2'(t) = \mu(1 - (x_1(t))^2)x_2(t) - x_1(t) \end{cases} \tag{20}$$

We have tested through this example the behaviour of VNODE-LP and CAPD when this ODE is simulated until a given time. We set $\mu = 4$ and start by simulating the ODE with scalars as initial conditions : $x_1(0) = 2$ and $x_2(0) = 0$. The simulation is performed until $t = 200$ is reached.

We show the results of the simulation for VNODE-LP and CAPD (Fig. 4.1). As we saw, ODE solvers compute dynamic time step in order to simulate ODEs. For simulation to $t = 200$, VNODE-LP performed 1838 different steps for a total computation time of 2.20 secondes. For CAPD, 2809 steps have been performed for a computation time of 2.28 secondes. This preliminary test shows that CAPD performs more steps in the same computation time as VNODE-LP and moreover it computes the derivative as well.

This test was exactly one of the normal uses of these solvers because initial conditions were scalars. But in ordinary differential equation based constraints, initial conditions are more likely intervals in real problems. We perform then a second experiment to see the behaviour of solutions during simulation for both solvers when intervals occur. This example has been chosen because it returns that solvers are not good at integrate it for initial interval conditions that are too large. Empirically found, the intervals used for simulation can not be larger than $1e^{-7}$.

Figure 2 shows the evolution of the diameter max of the solution computed at each time step. Simulation has been carried on until $t = 100$ for initial conditions close to the first simulation shown: Here $x_1(0) = 2 + \left[-7e^{-6}, 7e^{-6}\right]$ and $x_2(0) = 0 + \left[-7e^{-6}, 7e^{-6}\right]$ We see clearly that CAPD can manage better the overestimation at the begining of the simulation to then be as "bad" as VNODE-LP at the end.

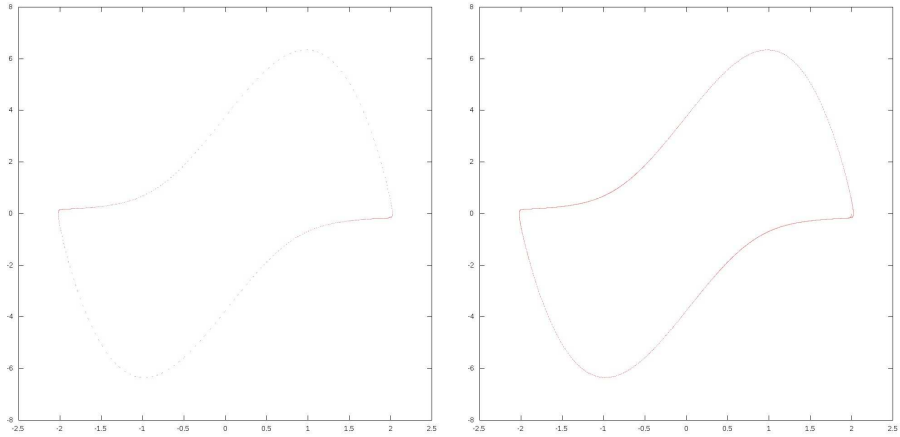

Fig. 1. Simulation of the Van Der Pol oscillator by VNODE-LP (left) and CAPD (right)

4.2 Parameter Estimation Problems

We consider the ODE as follows :

$$\mathbf{x}'(t) = A\mathbf{x}(t) \text{ with } A = \begin{pmatrix} 0.1 & 1 \\ -1 & 0.1 \end{pmatrix} \tag{21}$$

that is equivalent to :

$$\begin{cases} x_1'(t) = 0.1x_1(t) + x_2(t) \\ x_2'(t) = -x_1(t) + 0.1x_2(t) \end{cases} \tag{22}$$

This ODE describes a spiral unrolling toward infinity (as seen in Figure 3).

A parameter estimation problem involving (21) could be as this one: We look for the initial conditions $\mathbf{x}(0) \in \mathbb{R}^2$ for which the trajectories of the ODE belong to a given box $[\mathbf{m}_i]$ at time t_i (as shown with (16)).

As the ODE is linear, its trajectories can be formally computed using the matrix exponential:

$$\mathbf{x}(t) = e^{tA}\mathbf{x}(0) \tag{23}$$

Measurements can be taken using (23): $\mathbf{m}_i = e^{t_i A}\mathbf{x}(0)$ where we have fixed $\mathbf{x}(0)$. We define $[\mathbf{m}_i] = \mathbf{m}_i \pm 0.25$. Each constraint $\Phi(\mathbf{x}, t_i) \in [\mathbf{m}_i]$ is equivalent to finding the parallelepiped $\{\left(e^{t_i A}\right)^{-1}\mathbf{u} : \mathbf{u} \in \mathbf{m}_i \pm 0.25\}$. We perform a resolution process using values got by a preliminary simulation of the ODE (shown in Table 4.2).

Because this problem is a conjunction of inequality constraints, Branch and Prune using the unidimensional interval Newton method as filtering algorithm is applied. Figure 4 shows the results of this problem. Green boxes belong to the solution boxes $\mathcal{S}$ when red ones belong to the boundary boxes $\mathcal{B}$

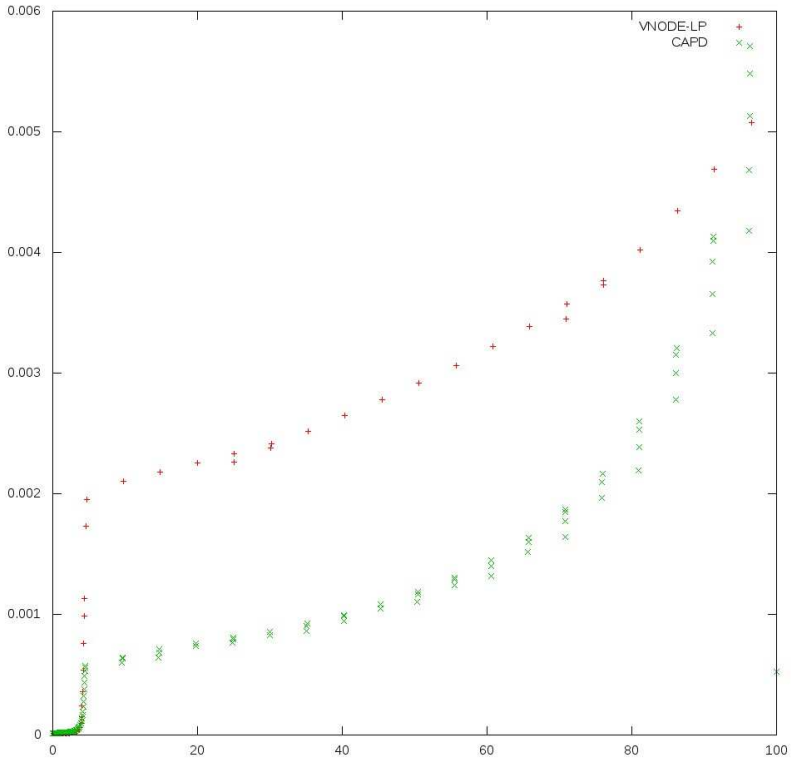

Fig. 2. Comparison of the maximal diameter evolution on VNODE-LP and CAPD

t_i	2	4	6	8
$\mathbf{m}_i$	(0.602335, -1.6189)	(-2.10414, 0.153895)	(1.24042, 2.25867)	(1.87804, -2.52567)

Table 1. Measurements used for a parameter estimation problem.

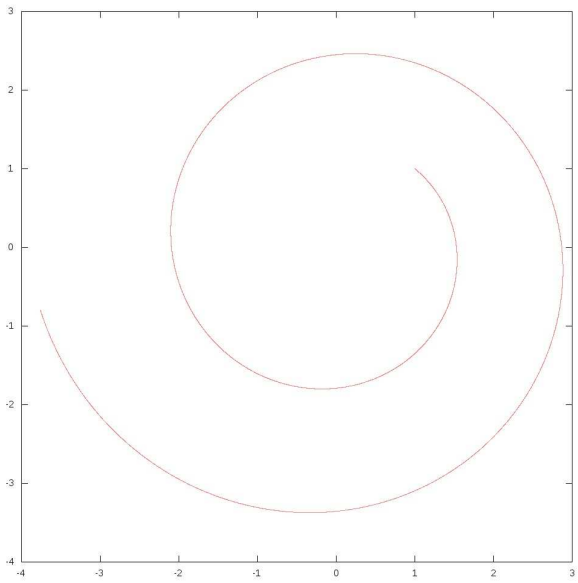

Fig. 3. Simulation of the ODE (21) for initial conditions (1, 1).

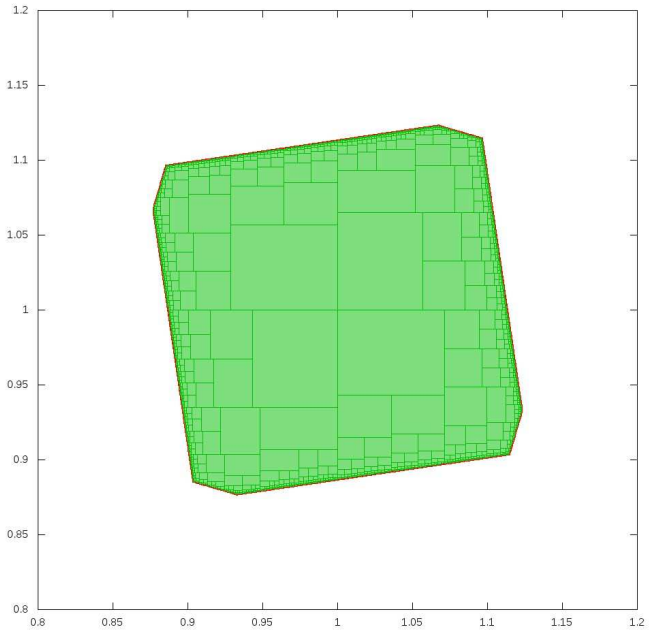

Fig. 4. Results for the spiral parameter estimation problem

4.3 Two-point Boundary Value Problems

We define an academic example of two-point boundary value problem. Let a second order ODE be :

$$y''(t) = -\lambda(1+y)\ t = [0,1] \tag{24}$$

which can be rewritten as this first order ODE :

$$\begin{cases} x_1'(t) = x_2(t) \\ x_2'(t) = -\lambda(1+x_1(t)) \end{cases},\ t = [0,1] \tag{25}$$

The two-point boundary problem P is to find $x_2(0) \in \left[x_2^0\right]$ and $x_2(1) \in \left[x_2^1\right]$ such that $x_1(0) = 0$ and $x_1(1) = 0$ or, introducing the ODE solution operator : Let $x_2(0) \in \left[x_2^0\right]$ and $x_2(1) \in \left[x_2^1\right]$,

$$(x_2(0), x_2(1)) \in \mathrm{Sol}(P) \Leftrightarrow \Phi(0, x_2(0)) - \begin{pmatrix} 0 \\ x_2(1) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{26}$$

Setting $\left[x_2^0\right] = [0,20]$ and $\left[x_2^1\right] = [-20,0]$, it took 1.3 secondes for the method to find the two solutions shown in Table 4.3. And most of all the solutions have been proven by the mulidimensional Newton method.

Solution enclosure	Width
1.036609890[4386868,7934569]	3.547701e-10
16.06565187[24917276,33426986]	8,50971e-10

Table 2. Solutions for $x_2(0)$for the two-point boundary value problem

Remark 8. The simulation of the ODE for the initial intervals was impossible for CAPD. In order to solve it, we must perform some bisections of the box. It shows the default of ODE solvers that are not always computing simulations when initial intervals are too large.

5 Conclusion

We have seen a method that allow to solve homogeneously problems with constraints involving ODEs. This method make this kind of problem being declared and processes in the NCSP framework. Advantages as the separation of the resolution process and its declaration would allow to work more generally on different problems that have until now only dedicated methods. However, the spine of this method is the integration of the ODE made by ODE solvers. We have seen that these solvers can easily calculate sharp enclosures of solutions but therefore, if the initial conditions are large interval or when a difficult ODE occurs (like for stiff ODE), this method shows its limits.

This method will benefit from improvements made both in the NCSP framework and the ODE solving. Future works should be turned to the ODE solver and certainly in the progress that could be made for larger interval initial conditions. And also with all improvements possible from the NCSP framework will make this method better. This method has been described also in a paper that will appear at CP 2010 conference[8]. That will allow to obtain a certain visibility and feedbacks in order to improve this method as well.

6 Aknowledgements

This work gave me the great opportunity to be intern in the National Institute of Informatic in Tokyo. I would like to thank all people here for there kindness, especially my supervisor professor Hosobe. I thanks also my supervisor doctor Goldsztejn in France for his constant availability, even if thousands of kilometers separated us. And finally a special thank to professor Wilczak, Auer and Nedialkov for their help during the early uses of their respective ODE solvers.

References

1. Ekaterina Auer, Andreas Rauh, Eberhard P. Hofer, and Wolfram Luther. Validated modeling of mechanical systems with smartmobile: Improvement of performance by valencia-ivp. In Peter Hertling, Christoph M. Hoffmann, Wolfram Luther, and Nathalie Revol, editors, *Reliable Implementation of Real Number Algorithms*, volume 5045 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2008.
2. M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4):361–369, 1998.
3. Maciej Capinski, Zbigniew Galias, Tomasz Kapela, Marian Mrozek, Paweł Pilarczyk, Daniel Wilczak, and Piotr Zgliczyński. The computer assisted proofs in dynamics group : http://capd.ii.uj.edu.pl/.
4. Cruz and Barahona. Constraint satisfaction differential problems. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.
5. Jorge Cruz and Pedro Barahona. Constraint reasoning in deep biomedical models. *Artificial Intelligence in Medicine*, 34(1):77–88, 2005.
6. Ernest Davis. Constraint propagation with interval labels. *Artif. Intell.*, 32(3):281–331, 1987.
7. P. Eijgenraam. *The Solution of Initial Value Problems Using Interval Arithmetic*. Mathematical Centre Tracts No. 144. Stichting Mathematisch Centrum, Amsterdam, 1981.
8. A. Goldsztejn, O. Mullier, D. Eveillard, and H. Hosobe. Including ordinary differential equations based constraints in the standard cp framework. In *CP 2010 conference (to appear)*, 2010.
9. Laurent Granvilliers. A symbolic-numerical branch and prune algorithm for solving non-linear polynomial systems. *J. UCS*, 4(2):125–146, 1998.
10. Laurent Granvilliers, Jorge Cruz, and Pedro Barahona. Parameter estimation using interval computations. *SIAM J. Scientific Computing*, 26(2):591–612, 2004.

11. L. W. Jackson. Interval arithmetic error-bounding algorithms. *SIAM J. Numer. Anal.*, 12(2):223–238, 1975.
12. Tomasz Kapela and Carles Simó. Computer assisted proofs for nonsymmetric planar choreographies and for stability of the eight. *Nonlinearity*, 20(5):1241, 2007.
13. Tomasz Kapela and Piotr Zgliczynski. An existence of simple choreographies for N-body problem - a computer assisted proof. April 25 2003. Comment: 19 pages, 9 figures.
14. O. Knüppel. Profil/bias - a fast interval library. *Computing*, 53(3-4):277–287, 1994.
15. F. Krückeberg. Ordinary differential equations. In Eldon Hansen, editor, *Topics in Interval Analysis*. Clarendon Press, Oxford, 1969.
16. R. E. Moore. Automatic error analysis in digital computation. Technical Report Space Div. Report LMSD84821, Lockheed Missiles and Space Co., Sunnyvale, CA, USA, 1959.
17. Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.
18. N. S. Nedialkov. Interval tools for odes and daes. In *SCAN '06: Proceedings of the 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, IEEE Computer Society, 2006.
19. Nedialko S. Nedialkov and Kenneth R. Jackson. The design and implementation of an object-oriented validated ODE solver. Technical report, October 21 2001.
20. Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.
21. Olavi Nevanlinna. Linear acceleration of Picard-Lindelöf iteration. *Numerische Mathematik*, 57(2):147–156, April 1990.
22. Tarek Raïssi, Nacim Ramdani, and Yves Candau. Set membership state and parameter estimation for systems described by nonlinear differential equations. *Automatica*, 40(10):1771–1777, 2004.
23. Andreas Rauh, Ekaterina Auer, and Eberhard P. Hofer. A novel interval method for validating state enclosures of the solution of initial value problems. February 27 2008.
24. Robert Rihm. On a class of enclosure methods for initial value problems. *Computing*, 53:369–377, 1994.
25. B. van der Pol. On relaxation oscillations. *Phil. Mag.*, 2:978–992, 1926.
26. Nobito Yamamoto. A numerical verification method for solutions of boundary value problems with local uniqueness by Banach's fixed-point theorem. *SIAM J. Numer. Anal.*, 35:2004–2013, 1998.